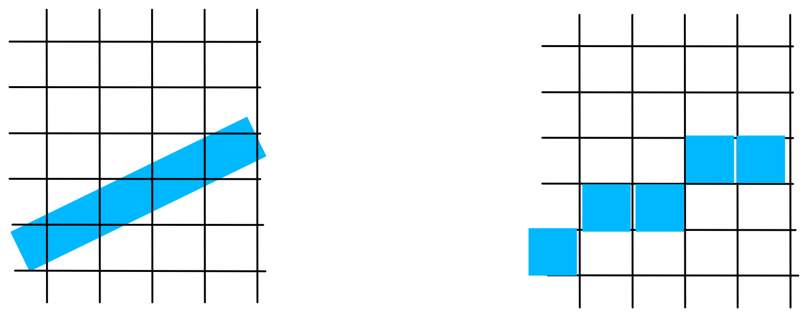
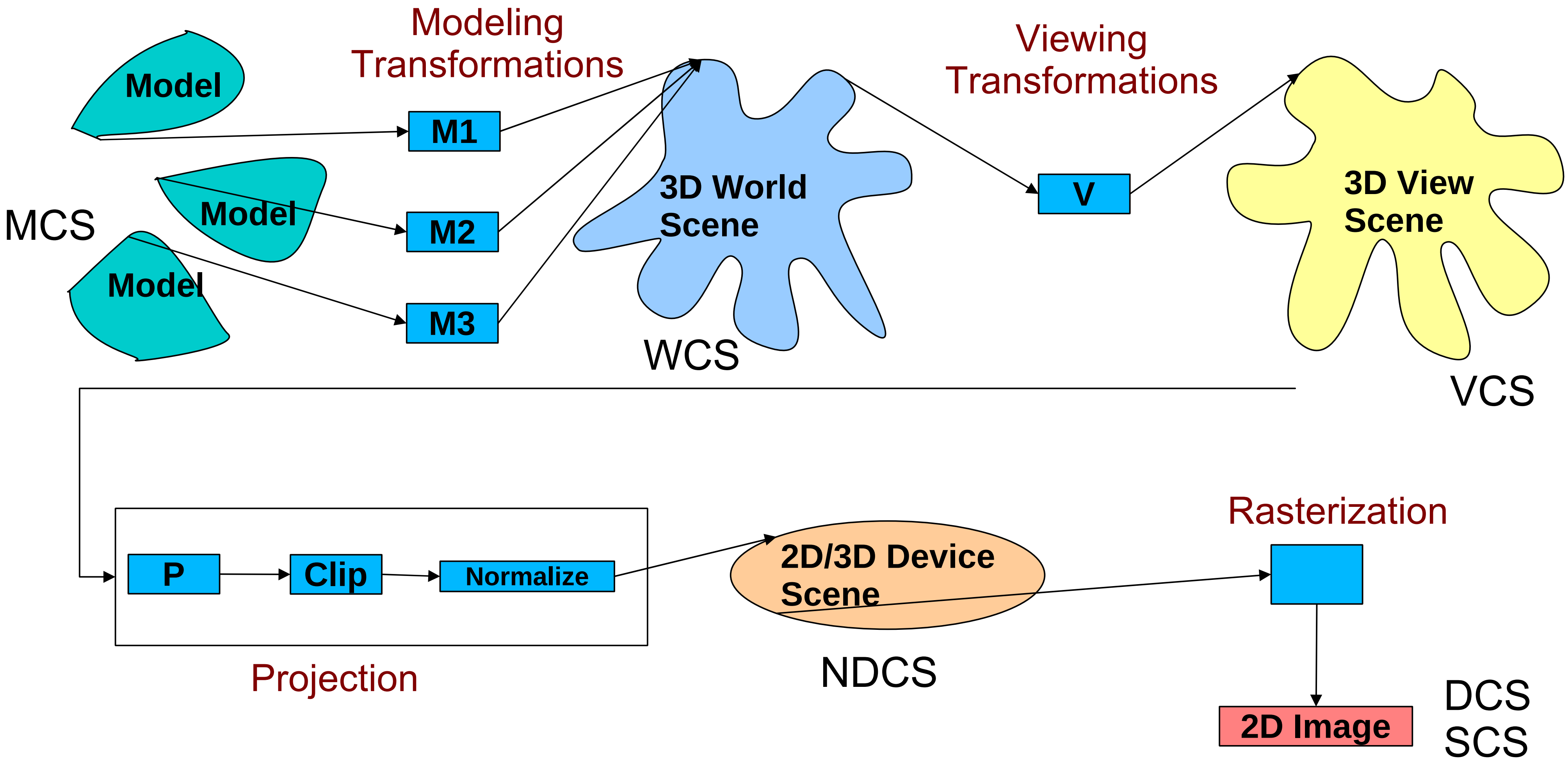
OUTPUT PRIMITIVES



CEng 477
Introduction to Computer Graphics
METU, 2007

Recap: The basic forward projection pipeline:



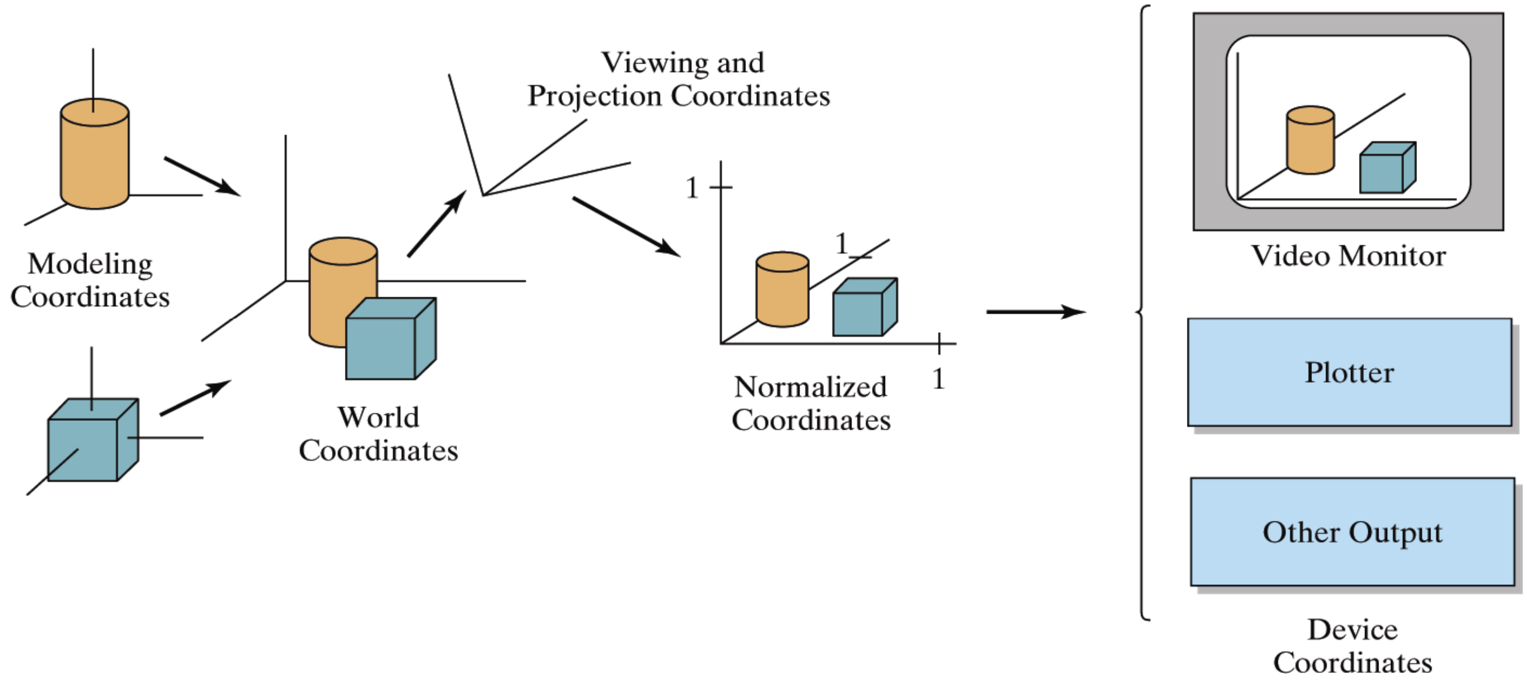


Figure 2-60

The transformation sequence from modeling coordinates to device coordinates for a three-dimensional scene. Object shapes can be individually defined in modeling-coordinate reference systems. Then the shapes are positioned within the world-coordinate scene. Next, world-coordinate specifications are transformed through the viewing pipeline to viewing and projection coordinates and then to normalized coordinates. At the final step, individual device drivers transfer the normalized-coordinaterepresentation of the scene to the output devices for display.

Screen vs. World coordinate systems

- Objects positions are specified in a Cartesian coordinate system called World Coordinate System which can be three dimensional and real-valued.
- Locations on a video monitor are referenced in **integer** *screen coordinates*. Therefore object definitions has to be scan converted to discrete screen coordinate locations to be viewed on a video monitor.

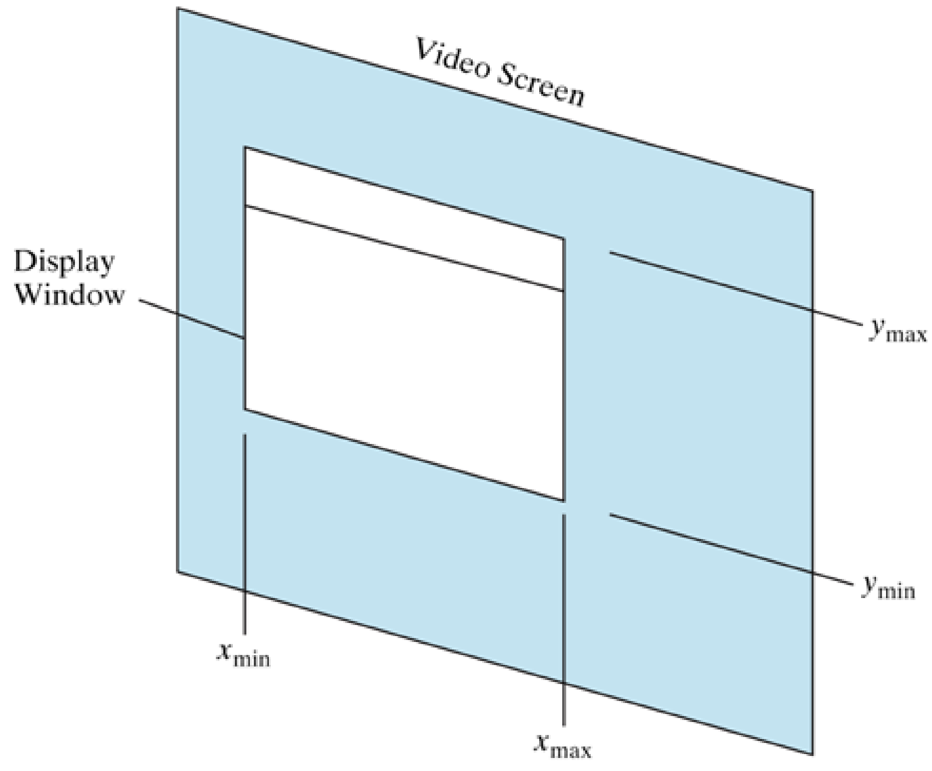
Specification of a 2D WCS in OpenGL

- *glMatrixMode (GL_PROJECTION);*

  glLoadIdentity ();

  gluOrtho2D (xmin, xmax, ymin, ymax);

- Objects that are specified within these coordinate limits will be displayed within the OpenGL window.



Output Primitives

- Graphic SW and HW provide subroutines to describe a scene in terms of basic geometric structures called output primitives.
- Output primitives are combined to form complex structures
- Simplest primitives
  - Point (pixel)
  - Line segment

Scan Conversion

- Converting output primitives into frame buffer updates. Choose which pixels contain which intensity value.
- Constraints
  - Straight lines should appear as a straight line
  - primitives should start and end accurately
  - Primitives should have a consistent brightness along their length
  - They should be drawn rapidly

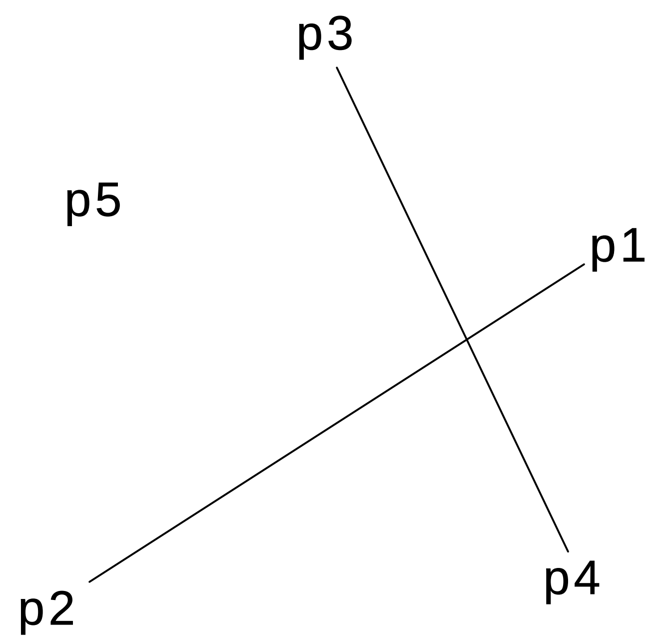
OpenGL Point Functions

```
• glBegin (GL_POINTS);
      glVertex2i(50, 100);
      glVertex2i(75, 150);
      glVertex2i(100, 200);
  glEnd();
```

OpenGL Line Functions

- *glBegin (GL_LINES);*

```
    glVertex2iv(p1);
    glVertex2iv(p2);
    glVertex2iv(p3);
    glVertex2iv(p4);
    glVertex2iv(p5);
glEnd();
```



OpenGL Line Functions

- *glBegin (GL_LINE_STRIP);*

    glVertex2iv(p1);

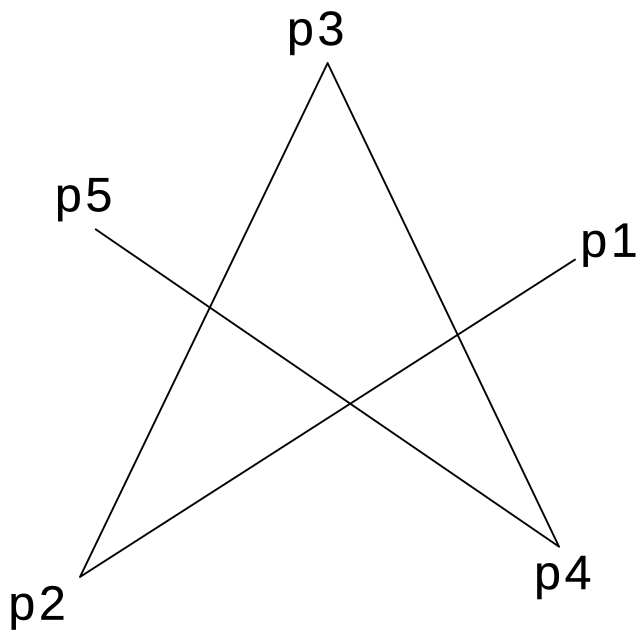
    glVertex2iv(p2);

    glVertex2iv(p3);

    glVertex2iv(p4);

    glVertex2iv(p5);

  glEnd();



OpenGL Line Functions

- *glBegin (GL_LINE_LOOP);*

    glVertex2iv(p1);

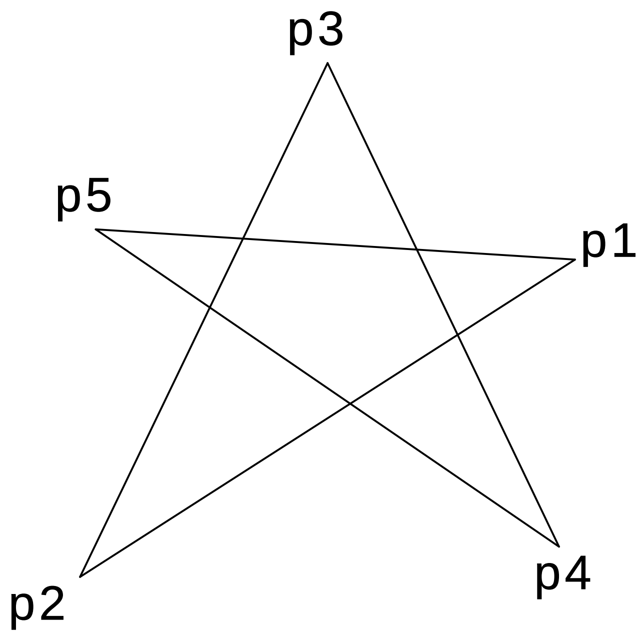
    glVertex2iv(p2);

    glVertex2iv(p3);

    glVertex2iv(p4);

    glVertex2iv(p5);

  glEnd();



Line Drawing Algorithms

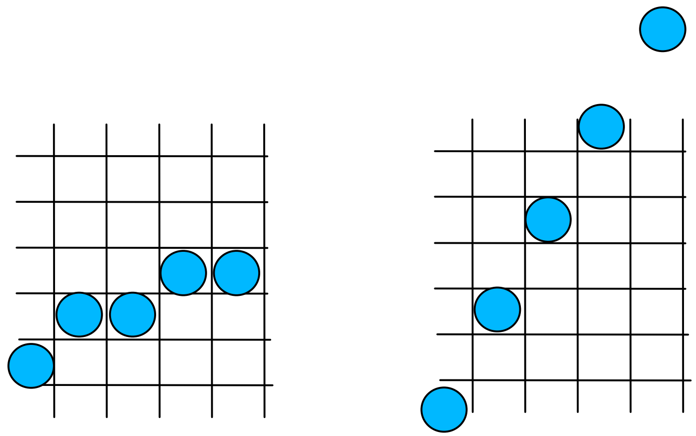
- Simple approach:

sample a line at discrete positions at one coordinate from start point to end point, calculate the other coordinate value from line equation (*slope-intercept line equation*).

$$y = mx + b \quad \text{or}$$

$$x = \frac{1}{m} y - \frac{b}{m}$$

$$m = \frac{y_{end} - y_{start}}{x_{end} - x_{start}}$$



Is this correct?

If $m > 1$, increment y and find x

If $m \leq 1$, increment x and find y

Digital Differential Analyzer

- Simple approach: too many floating point operations and repeated calculations
- Calculate y_{k+1} from y_k for a Δx value

$$\Delta y = m \Delta x \quad y_{k+1} = y_k + m \quad for \quad \Delta x = 1,\ 0 < m < 1$$

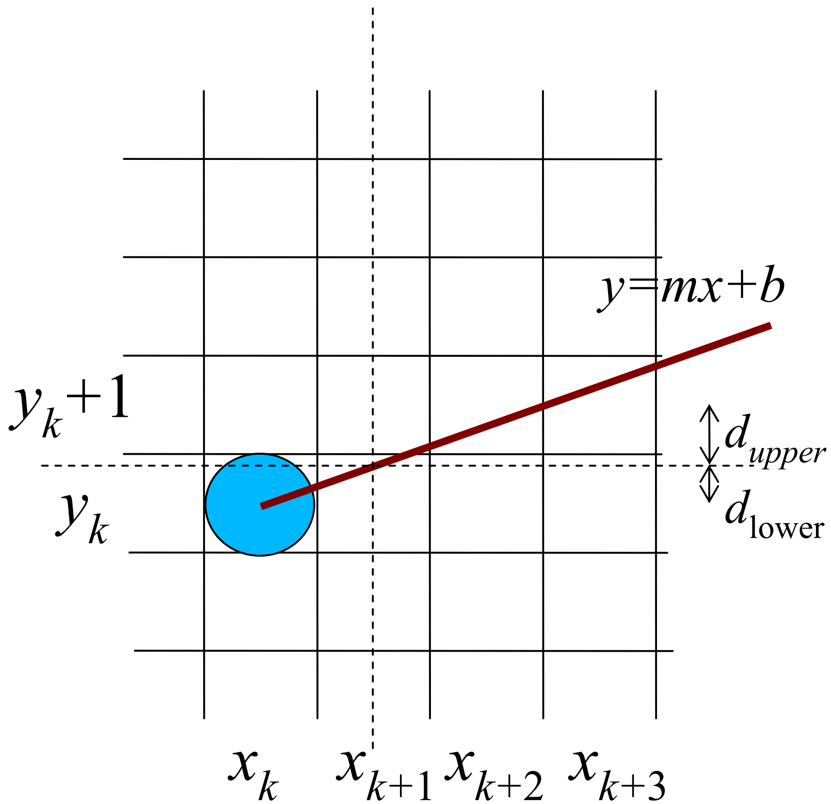
$$\Delta x = \frac{\Delta y}{m} \quad x_{k+1} = x_k + \frac{1}{m} \quad for \quad \Delta y = 1,\ m \geqslant 1$$

DDA

- Is faster than directly implementing $y=mx+b$. No floating point multiplications. We have floating point additions only at each step.
- But what about round-off errors?
- Can we get rid of floating point operations completely?

Bresenham's Line Algorithm

- DDA: Still floating point operations



Assume $|m| \leq 1$

If already at (x_k, y_k), choices:

(x_k+1, y_k) if $d_{lower} \leq d_{upper}$

(x_k+1, y_k+1) if $d_{lower} > d_{upper}$

$$y = m(x_k+1)+b \Rightarrow \begin{array}{l} d_{lower} = y - y_k = m(x_k+1)+b-y_k \\ d_{upper} = (y_k+1)-y = y_k+1-m(x_k+1)-b \end{array}$$

$$\Rightarrow d_{lower} - d_{dupper} = 2m(x_k+1)-2y_k+2b-1$$

$$m = \frac{\Delta y}{\Delta x} = \frac{y_{end}-y_{start}}{x_{end}-x_{start}}$$

define $p_k = \Delta x(d_{lower} - d_{upper}) = 2\Delta y x_k - 2\Delta x y_k + c$

$c = 2\Delta y + \Delta x(2b-1)$ independent from pixel position

$$\begin{array}{ll} \text{if } d_{lower} < d_{upper} \Rightarrow p_k < 0 \Rightarrow & \text{choose } y_k \\ \text{else} & \text{choose } y_k + 1 \end{array}$$

at step k+1:

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

$$x_{k+1} = x_k + 1 \Rightarrow p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

$(y_{k+1} - y_k)$: 0 if p_k was negative; 1 if p_k was positive

to calculate p_0 at the starting pixel position (x_0, y_0)

$$p_0 = 2\Delta y \cdot x_0 - 2\Delta x \cdot y_0 + c$$

$$c = 2\Delta y + \Delta x(2b - 1)$$

$$b = y_0 - \frac{\Delta y}{\Delta x} x_0 \Rightarrow c = 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x \Rightarrow p_0 = 2\Delta y - \Delta x$$

Bresenham's Line-Drawing Algorithm

Input: two line end points (x_0, y_0) and (x_{end}, y_{end})

draw (x_0, y_0)

$p_k \leftarrow 2\Delta y - \Delta x$; $x_k \leftarrow x_0$

while $x_k < x_{end}$

  $x_{k+1} \leftarrow x_k + 1$

  if $p_k \leq 0$ choose y_k

    $y_{k+1} \leftarrow y_k$; $p_{k+1} \leftarrow p_k + 2\Delta y$

  else choose $y_k + 1$

    $y_{k+1} \leftarrow y_k + 1$; $p_{k+1} \leftarrow p_k + 2\Delta y - 2\Delta x$

  draw (x_{k+1}, y_{k+1})

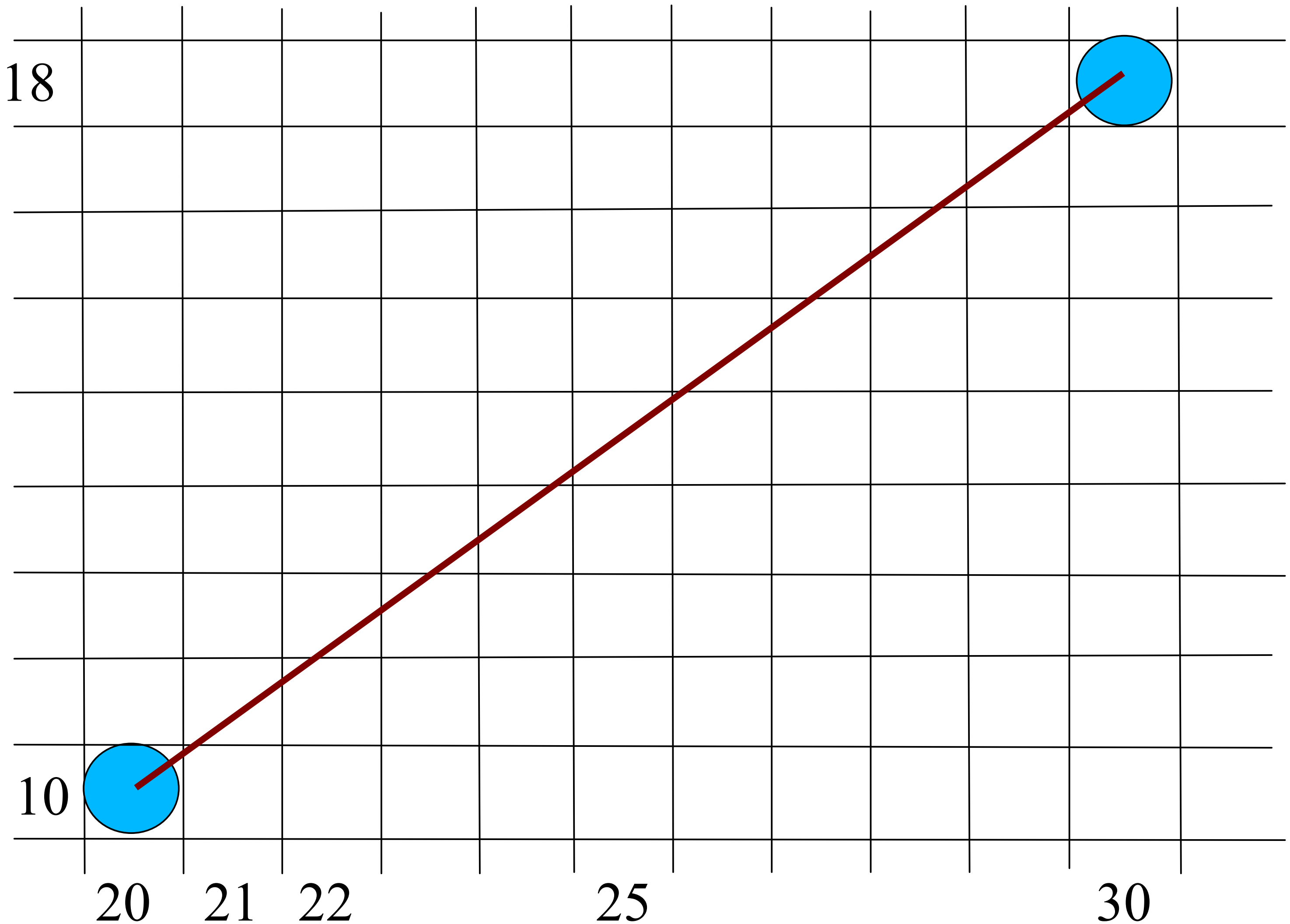
  $x_k \leftarrow x_{k+1}$

  $p_k \leftarrow p_{k+1}$

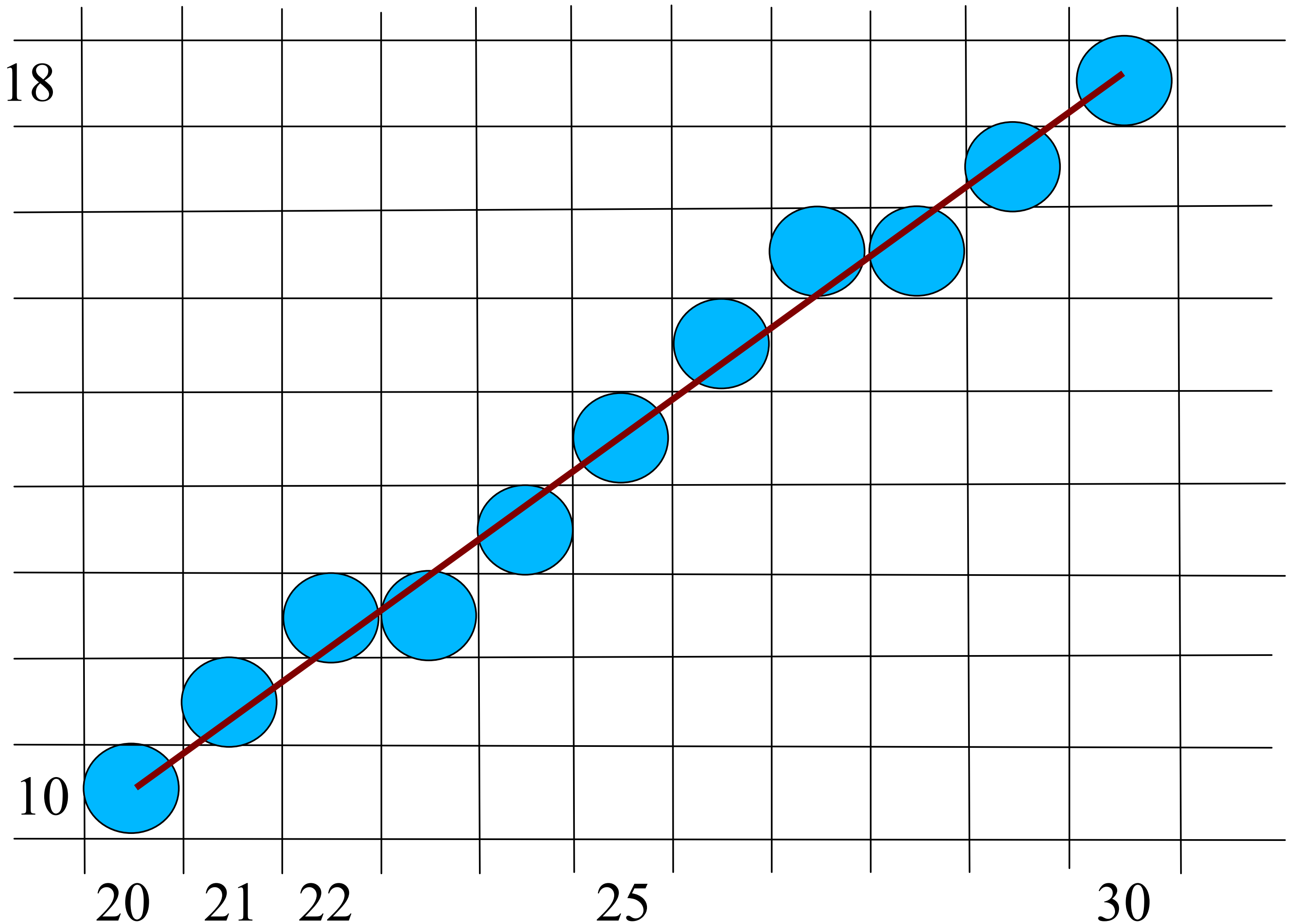
Example from the textbook

- Using Bresenham’s algorithm digitize the line with endpoints (20,10) and (30,18)

Example continued…



Plotted pixels



Circle Generation

- Circles can be approximated by a set of straight lines.

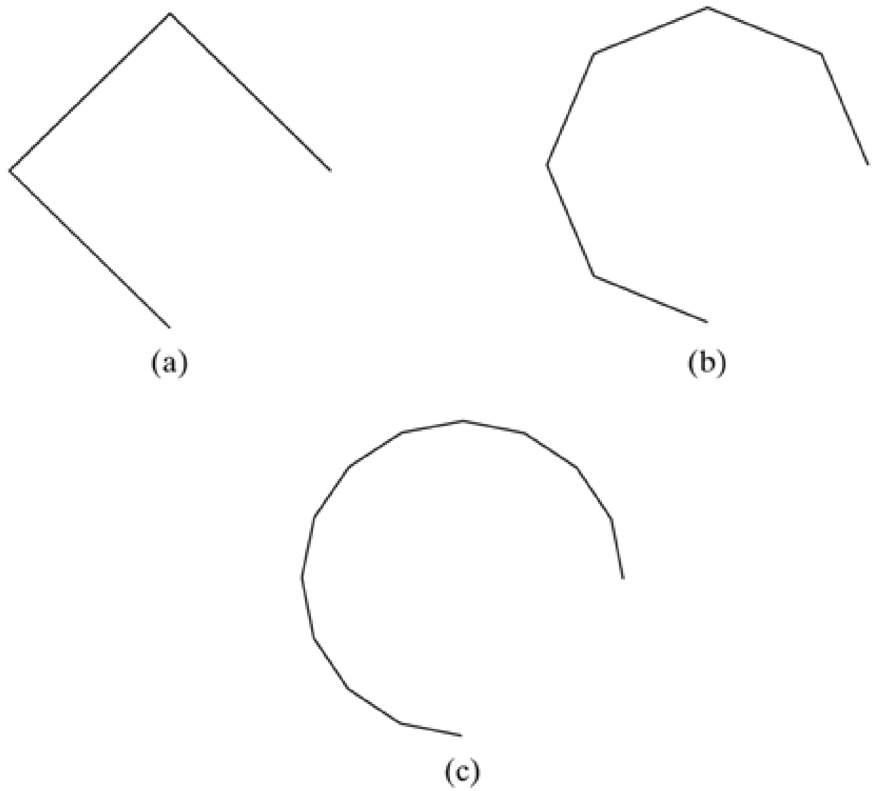


Figure 3-15

A circular arc approximated with (a) three straight-line segments, (b) six line segments, and (c) twelve line segments.

But, how many lines do we need for an acceptable representation?

How do we determine end points of lines?

Circle Drawing in OpenGL

- Routines for drawing circles or ellipses are **not included** in the OpenGL core library.
- GLU (OpenGL Utility) library has some routines for drawing spheres, cylinders, B-splines. Rational B-splines can be used to display circles and ellipses.

Circle Generation

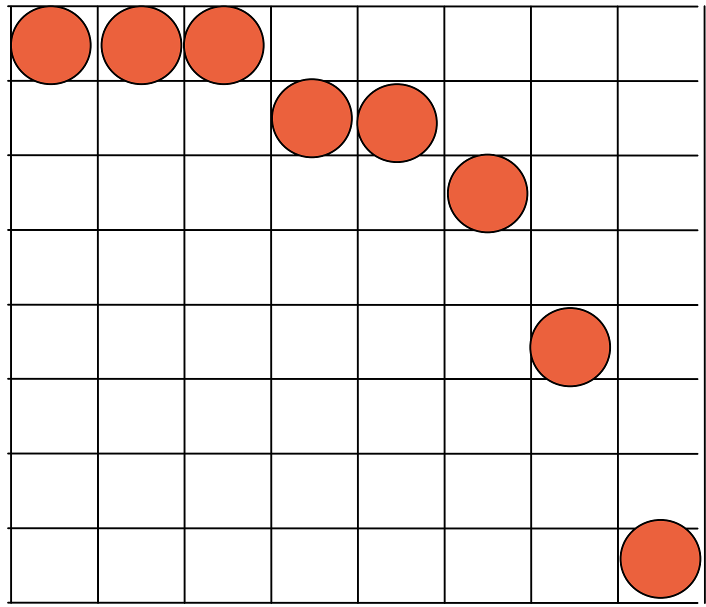
$$(x-x_0)^2+(y-y_0)^2=r^2$$

unit steps in $x \Rightarrow y=y_0 \mp \sqrt{r^2-(x-x_0)^2}$

- Computationally complex
- Non uniform spacing
- Polar coordinates:

$$x=r\cos(\theta)+x_c$$

$$y=r\sin(\theta)+y_c$$



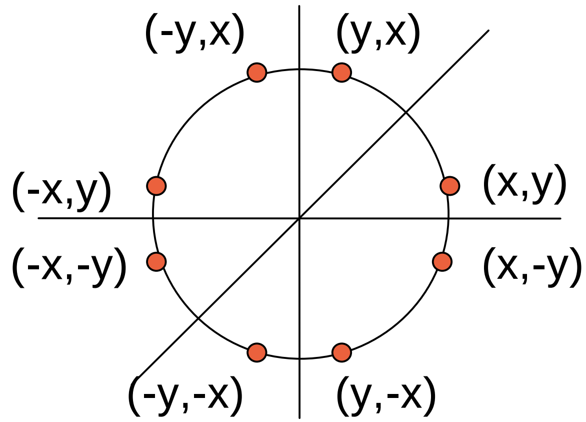
- Fixed angular step size to have equally spaced points

$$x_k = r\cos\theta \quad x_{k+1} = r\cos(\theta + d\theta)$$
$$y_k = r\sin\theta \quad y_{k+1} = r\sin(\theta + d\theta)$$

$$\begin{aligned}
x_{k+1} &= r\cos\theta\cos d\theta - r\sin\theta\sin d\theta \\
&= x_k\cos d\theta - y_k\sin d\theta \\
y_{k+1} &= r\sin\theta\cos d\theta + r\cos\theta\sin d\theta \\
&= y_k\cos d\theta + \boldsymbol{x_k}\sin d\theta
\end{aligned}$$

fixed $d\theta$ so compute $\cos d\theta$ and $\sin d\theta$ initially

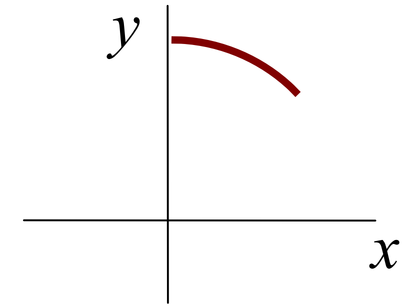
- Computation can be reduced by considering symmetry of circles:

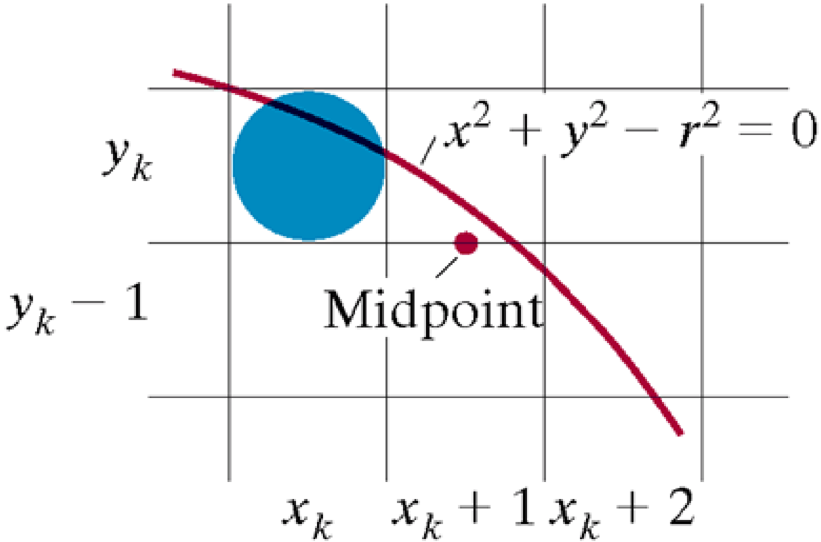


- Still too complex, multiplications, trigonometric calculations
- Bresenham's circle generation algorithm involves simple integer operations (comparing squares of pixel separation distances)
- Midpoint Circle Algorithm avoids squaring and generates the same pixels as Bresenhams’s algorithm.

Midpoint Circle Algorithm

- Consider the second octant. Increment x , decide on y





select which of 2 pixels, (x_k+1, y_k) or (x_k+1, y_k-1) are closer to the circle by evaluating the circle ction at the midpoint.

$$f(x,y) = x^2 + y^2 - r^2 \begin{cases} = 0 & \text{if on the circle choose } y_k - 1 \\ > 0 & \text{if outside the circle} \quad \text{choose } y_k - 1 \\ < 0 & \text{if inside the circle choose } y_k \end{cases}$$

$$p_k = f\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$p_{k+1} = f\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right) = (x_k + 1 + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

$$p_{k+1} - p_k = (x_k + 1 + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2 - (x_k + 1)^2 - \left(y_k - \frac{1}{2}\right)^2 + r^2$$

$$p_{k+1} = p_k + x_k^2 + 4x_k + 4 + y_{k+1}^2 - y_{k+1} + \frac{1}{4} - x_k^2 - 2x_k - 1 - y_k^2 + y_k - \frac{1}{4}$$

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

where y_{k+1} is either y_k or $y_k - 1$ depending on the sign of p_k.

if $p_k < 0$ p_{k+1} = $p_k + 2x_k + 3$

if $p_k \geq 0$ p_{k+1} = $p_k + 2x_k - 2y_k + 5$

computing p_0 at $(x_0, y_0) = (0, r)$

$$p_0 = f(1, r - \frac{1}{2})$$

$$= 1 + (r - \frac{1}{2})^2 - r^2$$

$$= \frac{5}{4} - r$$

if r is integer $p_0 = 1 - r$

Midpoint Circle Algorithm

Input: radius r and circle center (x_c, y_c)

draw$(0+x_c, r+y_c)$ (add x_c and y_c before plotting)

$p_k \leftarrow 1-r;\quad x_k \leftarrow 0;\ y_k \leftarrow r;$

while $x_k < y_k$

if $p_k < 0$ choose y_k

$y_{k+1} \leftarrow y_k;\quad p_{k+1} \leftarrow p_k + 2x_k + 3$

else choose $y_k - 1$

$y_{k+1} \leftarrow y_k - 1;\quad p_{k+1} \leftarrow p_k + 2x_k - 2y_k + 5$

$x_{k+1} \leftarrow x_k + 1$

draw $(x_{k+1} + x_c, y_{k+1} + y_c)$

$x_k \leftarrow x_{k+1};\quad y_k \leftarrow y_{k+1};$

$p_k \leftarrow p_{k+1}$

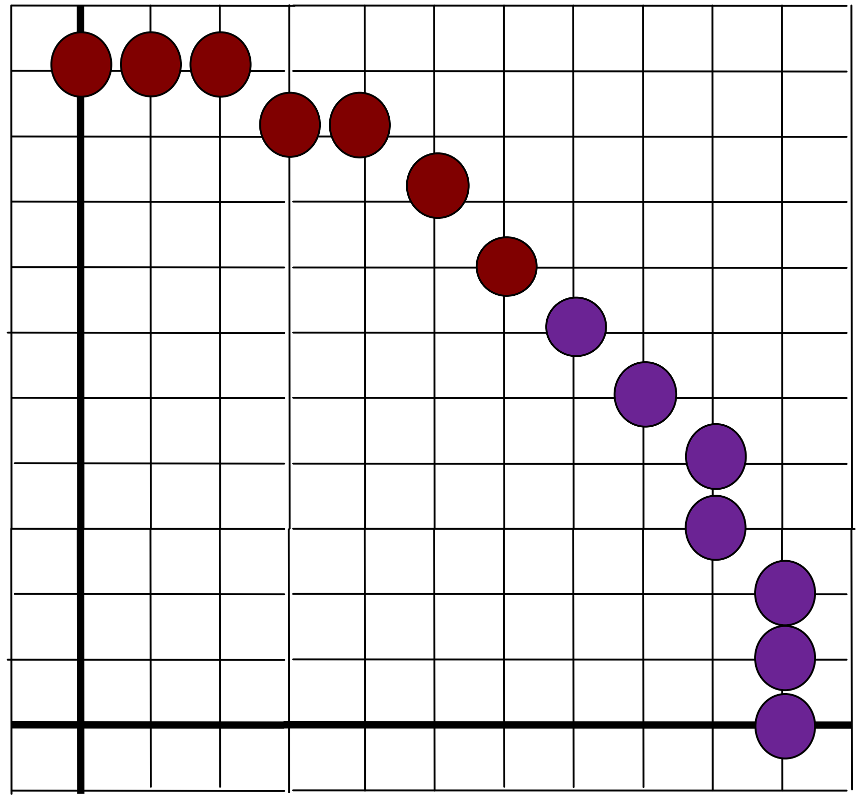
if $p_k < 0$ choose y_k

$$y_{k+1} \leftarrow y_k;\ \ p_{k+1} \leftarrow p_k + 2x_k + 3$$

else choose $y_k - 1$

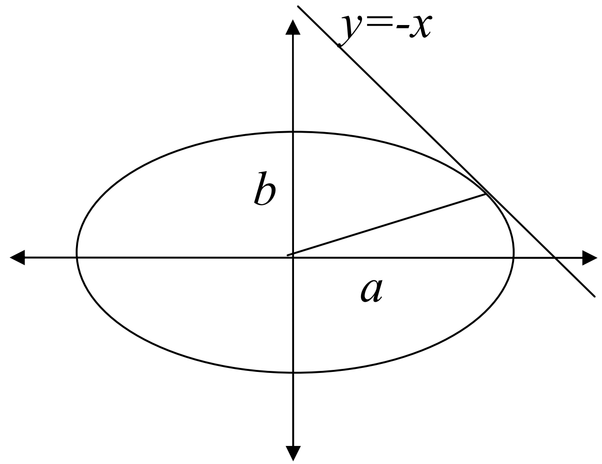
$$y_{k+1} \leftarrow y_k - 1;\ \ p_{k+1} \leftarrow p_k + 2x_k - 2y_k + 5$$

$x=0;\ \ y=0;\ \ r=10$		plot (0,10)
$p_k = 1 - 10 = -9$	choose y_k	plot (1,10)
$p_k = -9 + 2 + 3 = -4$	choose y_k	plot (2,10)
$p_k = -4 + 4 + 3 = 3$	choose y_k-1	plot (3,9)
$p_k = 3 + 6 - 18 + 5 = -4$	choose y_k	plot (4,9)
$p_k = -4 + 8 + 3 = 7$	choose y_k-1	plot (5,8)
$p_k = 7 + 10 - 16 + 5 = 6$	choose y_k-1	plot (6,7)
$p_k = 6 + 12 - 14 + 5 = 9$	choose y_k-1	plot (7,6)



Ellipse Generation

- Similar to circle generation with mid-point. Inside test.
- Different formula for points up to the tangent $y=-x$, slope<1.
  $(0,b)$ to tangent: increment x find y
  tangent to $(a,0)$: decrement y find x
- Mid-point algorithm is applicable to other polynomial equations:
  - Parabola, Hyperbola



$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$$